4

Systems
Optimization
Laboratory

A Parallel Decomposition Algorithm for Staircase Linear Programs

by
Robert Entriken

TECHNICAL REPORT SOL 88-21

December 1988

Department of Operations Research
Stanford University
Stanford, CA 94305

4

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-4022

A Parallel Decomposition Algorithm for Staircase Linear Programs

by
Robert Entriken

TECHNICAL REPORT SOL 88-21

December 1988

Research and reproduction of this report were partially supported by the National Science Foundation grants DMS-8800589 and ECS-8617905, the U.S. Department of Energy grant DE-FG03-87ER25028, the Office of Naval Research contract N00014-85-K-0343, and the Electric Power Research Institute contract RP-2940-1 at Stanford University, and the U.S. Air Force Office of Scientific Research, the U.S. Department of Energy contract DE-AC05-840R21400, the National Security Agency, and the Science Alliance Program of the State of Tennessee at Oak Ridge National Laboratory.

Table of Contents

A PARALLEL DECOMPOSITION ALGORITHM FOR STAIRCASE LINEAR PROGRAMS*

Robert Entriken

Department of Operations Research
Stanford University

Abstract

As part of an extended research project on the parallel decomposition of linear programs, a parallel algorithm for Staircase Linear Programs was designed and implemented. This class of problems encompasses a large range of planning problems and when decomposed has simple subproblem formulations and communication patterns. This makes its solution a manageable step toward our eventual goal of producing a general code that automatically exploits problem structures of various forms.

The results presented here were derived from an implementation for a Sequent Balance 8000 shared-memory multiprocessor. The algorithm itself is message-based but can run on either shared- or distributed-memory parallel computers.

A simple diet planning problem is used to demonstrate the principles of the algorithm's development and performance. When applied to this problem, the parallel decomposition algorithm shows promise relative to present serial optimization codes. The nonlinear optimization code MINOS 5.1 is used both as a basis for comparison and as a generic subproblem solver. The greatest room for speedup is in exploiting problem structures. The results show that decomposition can improve efficiency even with a single processor. Examples are given where multiple processors lead to still greater efficiency. (KR) $\leftarrow$

1. INTRODUCTION

The term Staircase Linear Program (SLP) describes a Linear Program (LP) that has a staircase pattern in the nonzero coefficients of its constraint matrix, as illustrated in Figure 1.1. Each "step" in the staircase typically corresponds to a collection of variables for a "period" of a planning horizon.

* This research was supported by the Electric Power Research Institute, the U.S. Air Force Office of Scientific Research, the U.S. Department of Energy, the National Security Agency, the National Science Foundation, the Science Alliance Program of the State of Tennessee, and the Department of Operations Research at Stanford University.

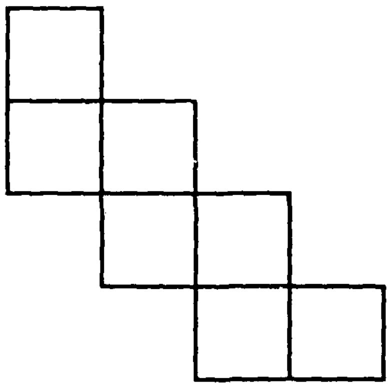

Figure 1.1. The classic staircase pattern.

The collections of variables are described as periods because the staircase structure arises most often from problems that represent systems over time [HL81]. In this report, we study multi-period or multi-day diet plans as examples of SLPs. The diet planning problem is a very simple LP that will help describe the ***reformulation*** and ***solution*** stages involved in solving staircase systems with a parallel computer.

To make proper use of a parallel computer, we must reformulate the original problem into multiple subproblems and then submit them to multiple processors as a means of obtaining greater throughput. The subproblems pass messages among themselves in a serial communication network of the form shown in Figure 1.2, where the circles represent the subproblems and the lines between them the communication paths. Each period's variables are associated only with the previous and following periods' variables in Figure 1.1. The serial structure results directly from the pattern of interdependencies between variables in the SLP. The medium of communication between processors in the parallel computer should have the ability to mimic a serial network.

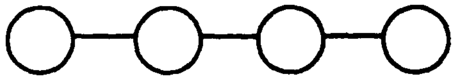

Figure 1.2. A serial communication network.

Abrahamson [Abr83] and Wittrock [Wit83] developed the topic of nested dual decomposition. The same material is repeated here for completeness, but in much less detail. Their work focused on the solution of such problems with a serial computer. We will consider here an extension to the use of a parallel computer, and paraphrase their results to prove the parallel algorithm converges.

Three major factors have been identified that significantly affect the speed and efficiency with which a solution is obtained in this framework:

(1) the *number of subproblems* into which we divide the SLP,

(2) the *number of processors* used to solve the subproblems, and

(3) the *order* in which the subproblems are *solved*.

A given subproblem may itself be a lower-dimensional SLP containing any number of *adjacent* steps of the original staircase.

It will be shown that there are diminishing returns associated with extensive decomposition of SLPs, and in the same way with increasing the number of processors used. The argument against both further decomposition and the use of more processors is the increasing cost of communication. It should be noted, however, that communication costs will diminish because of technology breakthroughs. Hence, this effect should be less significant in the future. Finally, with a better understanding of the dynamic and unpredictable path that the following parallel optimization algorithm takes to a solution, we will be better able to appreciate the subtle effect that the *solution order* of the subproblems has on overall performance.

2. THE FORMULATION OF SLPS

2.1. A One-Day Diet Problem

The One-Day Diet Problem is an example taken from Chvàtal [Chv83]. Its mathematical formulation can be found below as DIET1, with specific examples for the problem data A, b, c, and u. The problem is to find the optimal selection of six commodities* x, based on their corresponding costs (given by c), and their relative contributions toward satisfying the minimum daily requirements b, for CALCIUM, PROTEIN, and ENERGY. The number of requirements is limited to three for simplicity's sake, while the amount of each commodity selected to satisfy them is bounded above by satiation points u. The boxes in Figure 2.1 represent the pattern of nonzero coefficients in the costs and constraints. The problem is dense.

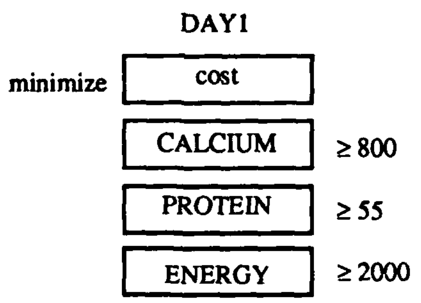

Figure 2.1. Structure of the one-day diet problem.

DIET1 is a linear program for determining a single day's purchases while spending the least amount of money. It will be used as a base case for formulating and studying multi-day diet-planning SLPs as examples. The primal and dual formulations of DIET1 are:

$$\text{(Primal) minimize } c^T x \text{ subject to } Ax \geq b, \quad 0 \leq x \leq u.$$
$$\text{(Dual) maximize } b^T \pi - u^T \sigma \text{ subject to } A^T \pi - \sigma \leq c, \quad \sigma \geq 0, \quad \pi \geq 0.$$

The following notation combines the dual variables π and σ with the primal formulation:

$$\begin{array}{lll} \text{(DIET1) minimize} & & c^T x \\ \text{subject to} & \pi: & Ax \geq b \\ & \sigma: & 0 \leq x \leq u, \end{array}$$

* OATMEAL, CHICKEN, EGGS, MILK, PIE, and PORK & BEANS.

where

$$A = \begin{pmatrix} 2 & 12 & 54 & 285 & 22 & 80 \\ 4 & 32 & 13 & 8 & 4 & 14 \\ 110 & 205 & 160 & 160 & 420 & 260 \end{pmatrix}, \qquad b = \begin{pmatrix} 800 \\ 55 \\ 2000 \end{pmatrix},$$

$$c^T = (3 \quad 24 \quad 13 \quad 9 \quad 20 \quad 19), \quad u = (4 \quad 3 \quad 2 \quad 8 \quad 2 \quad 2)^T.$$

The optimal selection of commodities is $\hat{x} = (4 \quad 0 \quad 0 \quad 4.5 \quad 2 \quad 0)^T$, with the corresponding dual solution $\hat{\pi} = (0 \quad 0 \quad 0.05625)^T$ and $\hat{\sigma} = (3.1875 \quad 0 \quad 0 \quad 0 \quad 3.625 \quad 0)^T$. Seven iterations of the simplex method [Dan63] are required to obtain this solution using MINOS 5.1 [MS87] with the default parameter settings, resulting in a minimum cost of $c^T\hat{x} = 92.5$. From the solution to the dual, one might determine that the ENERGY constraint is the most difficult to satisfy given the available commodities. Because there are zero prices on the CALCIUM and PROTEIN constraints, one can quickly determine that the commodities chosen are relatively low in ENERGY and high in CALCIUM and PROTEIN.

2.2. A Two-Day Diet Problem

DIET2 is a linear program that plans a selection of commodities over two days. It is similar to repeating DIET1 twice, but the daily requirement of ENERGY is relaxed to be satisfied in any combination over both days instead of each individually. The individual ENERGY constraints were added together, doubling the value of the right-hand side (RHS) entry.

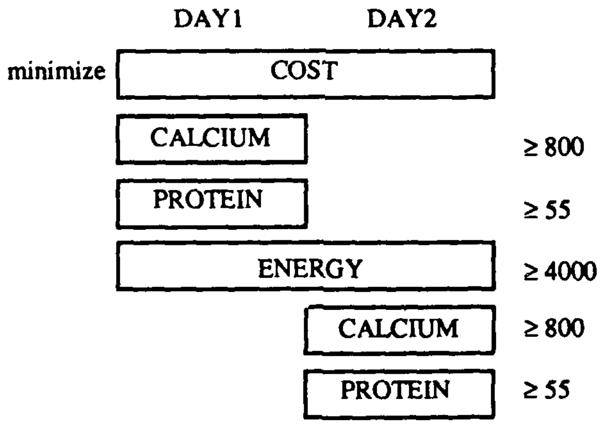

Figure 2.2. Structure of the two-day diet problem.

Figure 2.2 shows the 2-step sparse staircase coefficient pattern of DIET2. If the two individual DIET1-type ENERGY constraints had not been added together when forming DIET2, the optimal solution $\hat{x}$ of DIET1, repeated twice, would be the unique optimal solution to such a problem. However, because we combined the individual ENERGY constraints into a single constraint, this optimal solution to DIET2 is not unique, nor is it basic*.

* See Appendix A for a proof.

$$\begin{array}{llll} \text{(DIET2)} & \text{minimize} & & c_1^T x_1 + c_2^T x_2 \\ & \text{subject to} & \pi_1 : & A_1 x_1 \geq b_1 \\ & & \pi_2 : & B_1 x_1 + A_2 x_2 \geq b_2 \end{array}$$

$$\sigma_1 : \ 0 \leq x_1 \leq u_1, \quad \sigma_2 : \ 0 \leq x_2 \leq u_2,$$

where

$$A_1 = \begin{pmatrix} 2 & 12 & 54 & 285 & 22 & 80 \\ 4 & 32 & 13 & 8 & 4 & 14 \end{pmatrix}, \qquad b_1 = \begin{pmatrix} 800 \\ 55 \end{pmatrix},$$

$$B_1 = \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\ 2 & 12 & 54 & 285 & 22 & 80 \\ 4 & 32 & 13 & 8 & 4 & 14 \end{pmatrix},$$

$$b_2 = \begin{pmatrix} 4000 \\ 800 \\ 55 \end{pmatrix},$$

$$c_1^T = c_2^T = (3 \quad 24 \quad 13 \quad 9 \quad 20 \quad 19), \quad u_1 = u_2 = (4 \quad 3 \quad 2 \quad 8 \quad 2 \quad 2)^T.$$

The set of optimal solutions to DIET2 contains the optimal solution to DIET1 repeated twice. The locus of DIET2's optimal solutions is:

$$\begin{aligned} (\hat{x}_1^T \ \hat{x}_2^T) &= \lambda(4 \quad 0 \quad 0 \quad 5.125 \quad 2 \quad 0 \quad 4 \quad 0 \quad 0 \quad 3.875 \quad 2 \quad 0) + \\ &\quad (1-\lambda)(4 \quad 0 \quad 0 \quad 3.875 \quad 2 \quad 0 \quad 4 \quad 0 \quad 0 \quad 5.125 \quad 2 \quad 0), \quad \lambda \in [0,1], \\ (\hat{\pi}_1^T \ \hat{\pi}_2^T) &= (0 \quad 0 \quad 0.05625 \quad 0 \quad 0), \\ \hat{\sigma}_1^T = \hat{\sigma}_2^T &= (3.1875 \quad 0 \quad 0 \quad 0 \quad 3.625 \quad 0). \end{aligned}$$

The added freedom in choosing the two-day selection of goods allows selections of each day to be mutually dependent. Pair-wise dependence between repeating collections of variables is the characteristic of SLPs that gives them their serial communication structure. DIET2 is our example two-period SLP.

2.3. A Three-Day Diet Problem

Our example three-period SLP (DIET3) determines the optimal selection of three days' commodities. In this example the general staircase pattern begins to emerge from the nonzero coefficients of the constraints as shown in Figure 2.3. There are the first and last periods (DAY1 and DAY3) with only one adjacent period or collection of variables, and there is the middle period (DAY2) whose neighbors precede and follow it. For an n-day problem, there will be $n-2$ such "middle" periods, each of which has two neighbors.

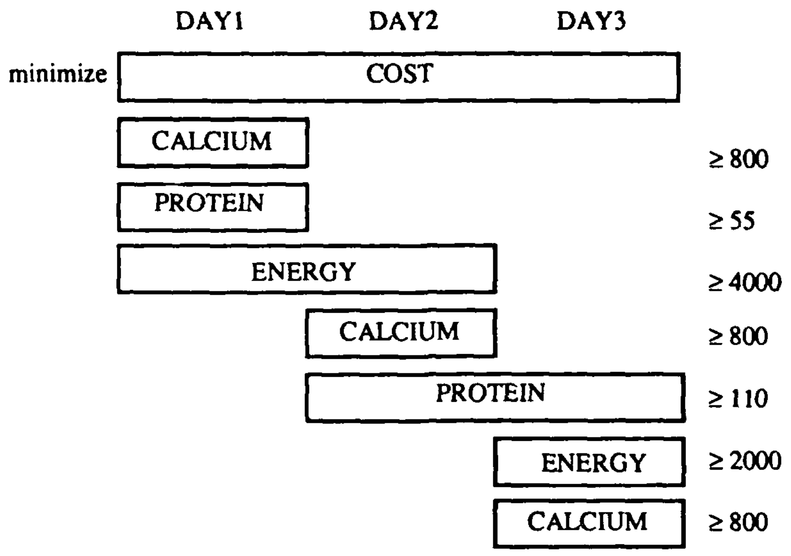

Figure 2.3. Structure of the three-day diet problem.

In DIET3 the ENERGY requirement is shared between the first two days as in DIET2. In addition, the PROTEIN requirement is shared between the last two days. This pattern can be propagated by extending the last constraint over two days and listing the remaining two individually below it. The third and fourth day share CALCIUM and after that the cyclical pattern repeats: ENERGY, PROTEIN, CALCIUM.

$$\begin{array}{llll} \text{(DIET3)} & \text{minimize} & c_1^T x_1 + c_2^T x_2 + c_3^T x_3 & \\ & \text{subject to} \quad \pi_1 : & A_1 x_1 & \geq b_1 \\ & \pi_2 : & B_1 x_1 + A_2 x_2 & \geq b_2 \\ & \pi_3 : & \qquad B_2 x_2 + A_3 x_3 & \geq b_3 \end{array}$$

$$\sigma_1 : \ 0 \leq x_1 \leq u_1, \quad \sigma_2 : \ 0 \leq x_2 \leq u_2, \quad \sigma_3 : \ 0 \leq x_3 \leq u_3,$$

where

$$A_1 = \begin{pmatrix} 2 & 12 & 54 & 285 & 22 & 80 \\ 4 & 32 & 13 & 8 & 4 & 14 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 800 \\ 55 \end{pmatrix},$$

$$B_1 = \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\ 2 & 12 & 54 & 285 & 22 & 80 \end{pmatrix},$$

$$b_2 = \begin{pmatrix} 4000 \\ 800 \end{pmatrix},$$

$$B_2 = \begin{pmatrix} 4 & 32 & 13 & 8 & 4 & 14 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 4 & 32 & 13 & 8 & 4 & 14 \\ 110 & 205 & 160 & 160 & 420 & 260 \\ 2 & 12 & 54 & 285 & 22 & 80 \end{pmatrix},$$

$$b_3 = \begin{pmatrix} 110 \\ 2000 \\ 800 \end{pmatrix},$$

$$c_i^T = (3 \quad 24 \quad 13 \quad 9 \quad 20 \quad 19), \quad u_i = (4 \quad 3 \quad 2 \quad 8 \quad 2 \quad 2)^T, \quad i = 1,2,3.$$

The solution to DIET3 is similar to that of DIET2 in that the optimal single-day selection from DIET1, repeated three times, *is* optimal overall. In addition, as before, this solution is one of a class of optimal solutions to DIET3:

$$\begin{aligned}
(\hat{x}_1^T\ \hat{x}_2^T) &= \lambda(4 \quad 0 \quad 0 \quad 5.125 \quad 2 \quad 0 \quad 4 \quad 0 \quad 0 \quad 3.875 \quad 2 \quad 0) + \\
&\quad (1-\lambda)(4 \quad 0 \quad 0 \quad 3.875 \quad 2 \quad 0 \quad 4 \quad 0 \quad 0 \quad 5.125 \quad 2 \quad 0), \quad \lambda \in [0,1], \\
\hat{x}_3^T &= (4 \quad 0 \quad 0 \quad 4.5 \quad 2 \quad 0), \\
(\hat{\pi}_1^T\ \hat{\pi}_2^T\ \hat{\pi}_3^T) &= (0 \quad 0 \quad 0.05625 \quad 0 \quad 0 \quad 0.05625 \quad 0), \\
\hat{\sigma}_1^T = \hat{\sigma}_2^T = \hat{\sigma}_3^T &= (3.1875, 0 \quad 0 \quad 0 \quad 3.625 \quad 0).
\end{aligned}$$

There is only one degree of freedom in the solution because the PROTEIN constraints are nonbinding.

3. REFORMULATING SLPS INTO MULTIPLE SUBPROBLEMS

We will now focus on reformulating the original general SLP into many interrelated subproblems using a technique called Benders decomposition [Ben62]. The purpose of creating a collection of subproblems in place of a single problem is to solve the collection simultaneously with a parallel computer. It will suffice for the scope of this report to present the subproblem formulations directly, and then go on to the parallel algorithm for solving the SLP. Each subproblem formulation contains *independent portions* of the original problem data, additional *necessary conditions* (cuts), and *accounting variables* that are simple machinery for algorithmic support—most calculations are implicit in the formulations, not explicit in the algorithm.

The following discussion will focus mainly on the case when the subproblems are solved to optimality, placing less emphasis on on cases when their solutions are infeasible or unbounded. This facilitates the exposition of the algorithm, saving the more complex cases for the next section when our insight is sufficiently developed.

3.1. Two-Period SLP

Benders decomposition differs from the more familiar Dantzig-Wolfe decomposition in that the former partitions an LP according to its variables whereas the latter partitions it according to its constraints. Each subproblem fixes the values of certain primal and dual variables of the original SLP in order to solve a reduced problem over a smaller set of variables. DIET2 can easily be decomposed into the two interdependent subproblems DIET2.1 and DIET2.2, that are solved one after the other, with appropriate modifications to certain algorithm parameters ($\tilde{\Pi}_2, \tilde{\delta}_2, \tilde{p}_2, \tilde{y}_1^k$ and $\tilde{\delta}_1^k$), until the modifications no longer affect the solutions of the two subproblems. These parameters are explained in detail following the formulations. In general, a subscript refers to a subproblem number, and a superscript k refers to a variable's value in the kth solution to the subproblem. We will use the notation $\tilde{x}$ for intermediate subproblem solutions, with $\hat{x}$ to denote the final solution for the full SLP. All other variables, primal and dual, will follow the same notation.

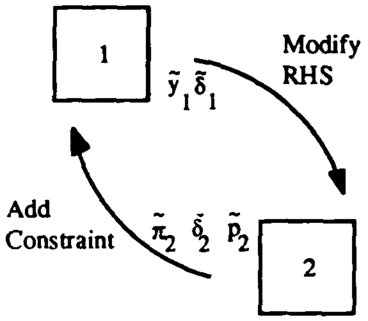
1
Modify RHS
$\tilde{y}_1 \tilde{\delta}_1$
Add Constraint
$\tilde{\pi}_2 \tilde{\delta}_2 \tilde{p}_2$
2

Figure 3.1. Information flow in the two-period SLP.

DIET2.1 initially drops the ENERGY constraint and the second day's PROTEIN and CALCIUM constraints from the original SLP, and solves only with regard to the first day's CALCIUM and PROTEIN requirements and minimum COSTS. Naturally, a certain amount of ENERGY is thereby offered by DIET2.1. It is calculated as $y_1 = B_1 x_1$ in the rows corresponding to the dual variables ρ_1.

$$\begin{array}{llrl}
\text{(DIET2.1)} & \text{minimize} & c_1^T x_1 + \theta_1 & = z_1 \\
\text{subject to} & \pi_1 : & A_1 x_1 & \geq b_1 \\
 & \rho_1 : & B_1 x_1 - I y_1 & = 0 \\
 & \mu_1 : & \tilde{\Pi}_2 y_1 + \tilde{\delta}_2 \theta_1 & \geq \tilde{p}_2
\end{array}$$

$$\sigma_1 : \ 0 \leq x_1 \leq u_1.$$

$$\begin{array}{llrl}
\text{(DIET2.2)} & \text{minimize} & c_2^T x_2 & = z_2 \\
\text{subject to} & \eta_2 : & \tilde{\delta}_1^k w_2 & = \tilde{\delta}_1^k \\
 & \pi_2 : & A_2 x_2 + \tilde{y}_1^k w_2 & \geq b_2
\end{array}$$

$$\sigma_2 : \ 0 \leq x_2 \leq u_2, \quad w_2 \geq 0, \quad \eta_2 = 0 \text{ if } \tilde{\delta}_1^k = 0.$$

Taking $(\tilde{\delta}_1^1, \tilde{z}_1^1, \tilde{x}_1^1, \tilde{y}_1^1, \tilde{\theta}_1^1)$ as the first solution to DIET2.1 with its parameters $\tilde{\Pi}_2, \tilde{\delta}_2$, and $\tilde{p}_2$ set to zero, we note that $\tilde{\delta}_1^1$ and $\tilde{y}_1^1$ will be used in DIET2.2's formulation. We set $\tilde{\delta}_1^1 = 0$ if DIET2.1 finishes unbounded, $\tilde{\delta}_1^1 = 1$ if it finishes optimal, and $\tilde{\delta}_1^1 =$ *undefined* if it finishes infeasible, because the entire SLP must be infeasible.

Let us assume that DIET2.1 finishes optimal. We set $\tilde{\delta}_1^1 = 1$ so that DIET2.2 adopts $\tilde{y}_1^1$ as the amount of ENERGY offered by DIET2.1. (Note that $\tilde{y}_1^1$ is effectively subtracted from the right-hand side when $\tilde{\delta}_1^1 = 1$ because w_2 is fixed at 1.) After solving DIET2.2 and assuming optimality, we return to DIET2.1 with the optimal prices on DIET2.2's constraints corresponding to π_2. These are used to impose a new constraint on y_1 to ensure that the same dual feasible extreme point $\tilde{\pi}_2^1$ will not be obtained again when DIET2.2 is solved, unless the optimal value of y_1 has been reached. This is the verbal interpretation of the μ_1 rows of the first-day subproblem.

The inequality $\tilde{\Pi}_2 y_1 + \tilde{\delta}_2\theta_1 \geq \tilde{p}_2$ in DIET2.1 is a collection of added constraints $(\tilde{\pi}_2^k)^T y_1 + \tilde{\delta}_2^k\theta_1 \geq \tilde{p}_2^k$ obtained from K dual solutions to the second-day subproblem $(\tilde{\delta}_2^k, \tilde{z}_2^k, \tilde{\eta}_2^k, \tilde{\pi}_2^k,\ k = 1, 2, \ldots, K)$. In particular, $\tilde{\delta}_2 = (\tilde{\delta}_2^1, \ldots, \tilde{\delta}_2^K)^T$ is a vector of Kronecker delta functions indicating optimality in each of the K solutions. If the k^{th} *dual* solution to DIET2.2 is dual feasible and bounded above, then we set $\tilde{\delta}_2^k = 1$; if it is dual feasible and unbounded above then $\tilde{\delta}_2^k = 0$; if dual infeasible (indicated by an unbounded primal solution), the two-period *primal* SLP is unbounded. The vector $\tilde{p}_2 = (\tilde{p}_2^1, \ldots, \tilde{p}_2^K)^T$ is a corresponding collection of scalars calculated as $\tilde{p}_2^k = \tilde{z}_2^k - \tilde{\eta}_2^k$ from $\tilde{z}_2^k$ (the objective value, or sum of infeasibilities) and $\tilde{\eta}_2^k$ (the price, or multiplier, on the constraint corresponding to η_2). Finally, $\tilde{\Pi}_2 = (\tilde{\pi}_2^1, \ldots, \tilde{\pi}_2^K)^T$, where $\tilde{\pi}_2^k$ is the vector of prices or multipliers on the constraints corresponding to π_2.

One way of interpreting the addition of constraints to DIET2.1 is that the matrix A_2 is being approximated by the independent rows in $\tilde{\Pi}_2$. It is therefore sufficient to carry along at most the number of constraints corresponding to the row-rank of A_2. In other words, when such constraints become slack, they may be discarded. However, if they turn out to be binding in the final optimal solution, they will be regenerated. Discarding cuts runs the risk of cycling due to degeneracy; it could lead to a repeated pattern of discarding and regenerating the same constraints.

3.2. n-Period SLP

The subproblems of the n-period SLP are of three types: the first period as in DIET2.1, the last period as in DIET2.2, and those with two adjacent subproblems, which are a combination of the two other forms. The three subproblems of DIET3 (DIET3.1, DIET3.2, DIET3.3) exemplify these three types and thereby those of an n-period SLP.

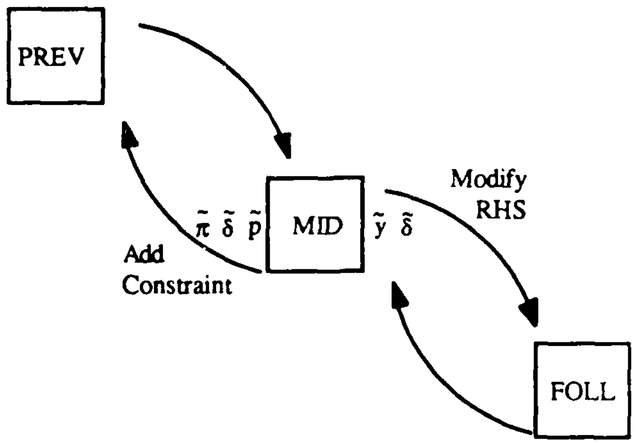
PREV
MID
FOLL
$\tilde{\pi}\ \tilde{\delta}\ \tilde{p}$
Add Constraint
$\tilde{y}\ \tilde{\delta}$
Modify RHS

Figure 3.2. Information flow in the n-period SLP.

Following the manner in which DIET2 was decomposed into two subproblems, we will quickly go through the division of the three-day diet problem into three subproblems. This exercise will demonstrate two key procedures. The first is the formulation of a subproblem that accepts solutions from two others, the previous and following subproblems, as opposed to only one other in the DIET2 cases; this allows a generalization to the n-period SLP. The second procedure is associated with the possibility of using more than one processor, thereby carrying Benders decomposition into the multiprocessor environment. As described, a subproblem may have more than one neighbor. Hence, an algorithmic choice must be made as to which neighbor to solve next when there is only one processor. When multiple processors are available, a choice need not be made—all of the neighboring subproblems may be solved simultaneously.

The three subproblems of DIET3 are formed by partitioning the daily selections as before. DIET3.1 will plan purchases for the first day, DIET3.2 for the second day, and DIET3.3 for the third day. As before, $\tilde{y}_1$ is the amount of ENERGY in the first day's selection, while $\tilde{y}_2$ is the amount of PROTEIN in the second day's selection.

$$\begin{array}{lrll}
\text{(DIET3.1)} & \text{minimize} & c_1^T x_1 \quad + \quad \theta_1 & = z_1 \\
 & \text{subject to} \quad \pi_1 : & A_1 x_1 & \geq b_1 \\
 & \rho_1 : & B_1 x_1 - I y_1 & = 0 \\
 & \mu_1 : & \tilde{\Pi}_2 y_1 + \tilde{\delta}_2 \theta_1 & \geq \tilde{p}_2
\end{array}$$

$$\sigma_1 : \ 0 \leq x_1 \leq u_1.$$

$$\begin{array}{lrll}
\text{(DIET3.2)} & \text{minimize} & c_2^T x_2 \quad + \quad \theta_2 & = z_2 \\
 & \text{subject to} \quad \eta_2 : & \tilde{\delta}_1^k w_2 & = \tilde{\delta}_1^k \\
 & \pi_2 : & A_2 x_2 \quad + \tilde{y}_1^k w_2 & \geq b_2 \\
 & \rho_2 : & B_2 x_2 - I y_2 & = 0 \\
 & \mu_2 : & \tilde{\Pi}_3 y_2 + \tilde{\delta}_3 \theta_2 & \geq \tilde{p}_3
\end{array}$$

$$\sigma_2 : \ 0 \leq x_2 \leq u_2, \quad w_2 \geq 0, \quad \eta_2 = 0 \text{ if } \tilde{\delta}_1^k = 0.$$

$$\begin{array}{lrll}
\text{(DIET3.3)} & \text{minimize} & c_3^T x_3 & = z_3 \\
 & \text{subject to} \quad \eta_3 : & \tilde{\delta}_2^k w_3 & = \tilde{\delta}_2^k \\
 & \pi_3 : & A_3 x_3 + \tilde{y}_2^k w_3 & \geq b_3
\end{array}$$

$$\sigma_3 : \ 0 \leq x_3 \leq u_3, \quad w_3 \geq 0, \quad \eta_3 = 0 \text{ if } \tilde{\delta}_2^k = 0.$$

Subproblem	**Parameters**
DIET3.1	$\tilde{\Pi}_2, \tilde{\delta}_2, \tilde{p}_2$
DIET3.2	$\tilde{\delta}_1^k, \tilde{y}_1^k, \tilde{\Pi}_3, \tilde{\delta}_3, \tilde{p}_3$
DIET3.3	$\tilde{\delta}_2^k, \tilde{y}_2^k$

Each of the subproblem formulations contains parameters that are based on the solutions of neighboring subproblems. The next section will describe the continual updating of these parameters as part of a parallel decomposition algorithm.

4. SOLVING SUBPROBLEMS ON A PARALLEL COMPUTER

From the subproblem formulations in the previous section, we have seen that any given subproblem contains from two to five parameters. These are initially undefined, and are first given values when a neighboring solution is communicated. In the serial dual-decomposition algorithm we begin by solving the last-period subproblem (knowing that any solution must meet the dual constraints corresponding to x_n), and then work toward obtaining a dual-feasible solution and then an optimal solution for all periods, if possible.

In the parallel algorithm we begin by solving all subproblems simultaneously, with their communication parameters initially set to zero. Their solutions, despite not including neighboring information, are still relevant and can be used to construct modifications. When used to modify right-hand sides, they will direct the new optimal solutions to a possibly different set of relevant solutions. When used to add constraints, the type of constraint depends on whether the neighboring subproblem solution was infeasible or feasible and, if feasible, whether bounded or unbounded. Initially only the last subproblem solution may be used to add an optimality constraint, yet any infeasible solution may be used to generate a necessary condition for feasibility (a feasibility constraint). In general, an optimality constraint may be passed only if the present subproblem already contains such a constraint (except the last subproblem, of course).

Just as a two-subproblem decomposition, with no middle subproblems, is a special case of an n-subproblem decomposition, the single-processor algorithm is a special case of the multiprocessor parallel algorithm with one subproblem being solved at a time instead of many. With only one processor, the parallel algorithm reduces to Benders *decomposition*.

The parallel computer architecture used for solving SLPs with the parallel algorithm is assumed to have numerous independent and powerful processors. The amount of memory available locally for the use of each individual processor is assumed to be substantial ($\geq \frac{1}{2}$ Megabyte) since the optimization code (MINOS 5.1) stored for each processor is large, and each subproblem data-set can be large. Shared- and distributed-memory multiprocessing computers as well as distributed-processing computers are suitable for our application. Table 4.1 gives some examples of commercially available processors.

Type	Name	Number of Processors	Size of Memory
Shared Memory	Sequent Balance 8000	8 NS32032	8 Meg.
Distributed Memory	NCUBE	4 Specialized	1/2 Meg ea.
	Intel iPSC hypercube	64 80286	1/2 Meg ea.
Distributed Processor	VAX cluster on Ethernet	5 VAXstation IIs	8 Meg ea.

Table 4.1. Examples of parallel computer architectures.

In a discussion of alternative architectures, the issues of computational and communication loads are of primary importance. The ideal is to distribute the computational load equally across all processors while keeping the time used for communication to a minimum. The reformulation (initialization) stage requires a large flow of information between the processors, but in the solution stage the messages are typically small and infrequent. This is evident because reformulation involves distributing the original data into n subproblems, whereas during the solution stage most time is spent solving LP subproblems with the simplex method.

The description of this algorithm is directed primarily toward a shared-memory implementation. Parallel processors with distributed memory require an additional scheme for distributing the work load so that the processors are responsible for disjoint subsets of the subproblems. If the subset of a processor contains more than one subproblem, they should be handled as in the single-processor shared-memory case (a special case of the parallel algorithm below). In addition, the scheduling of subproblems between processors becomes an implicit result of message passing.

4.1. Processes, Jobs and Queues

There will be a user-specified number of subproblems n and processors p involved in solving a general SLP. The number of processors could exceed the number of subproblems ($p > n$) but this would leave the extra processors unused, or inefficiently loaded. Hence, we assume that $p \leq n$.

Associated with each of the n subproblems is a job consisting of the loop of tasks in Figure 4.1. The term "job" is used to emphasize the fact that it encompasses more than solving linear program subproblems. The tasks of each job are repeated in succession and any processor can execute them. A job is always in one of three states: **run**, running; **pend**, waiting to be run; or **sleep**, solved and waiting for new information.

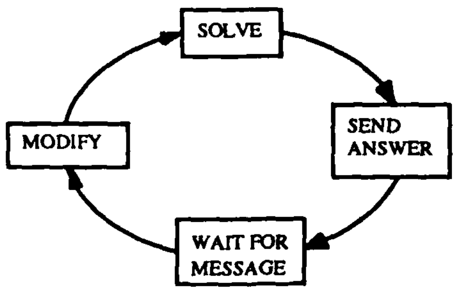

Figure 4.1. The RUN JOB loop.

The p processors repeatedly execute the loop in Figure 4.2, which transfers jobs among three shared queues (**run, pend,** and **sleep**) according to the result of running through the job loop. Each queue corresponds to one of the three job states.

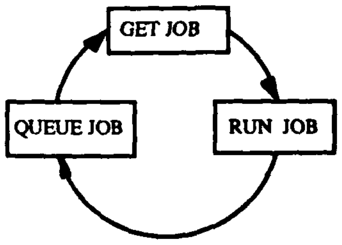

Figure 4.2. The process loop.

The **run** queue contains the jobs currently running on p or fewer processors, the results of which will define the next states of the jobs. Thus, it will never contain more than p jobs. The **pend** queue is where a processor finds jobs (subproblems) waiting to be run (solved), and the **sleep** queue is where jobs reside while at the WAIT FOR MESSAGE step.

In the process loop of Figure 4.2, GET JOB involves inspecting the **pend** queue and, if there is an available job, transferring it to the **run** queue. Jobs are run (RUN JOB) as a repeated sequence of tasks, beginning at the SOLVE step. After the jobs pass through the SOLVE step, they continue around the loop unless the solution is unchanged, in which case the job is placed in the **sleep** queue (QUEUE JOB) to wait for a message.

In a more advanced implementation, one might consider interrupting the SOLVE step after a specified number of iterations and placing the job, though unfinished, back in the **pend** queue. This would have the effect of further balancing the computation power across the subproblems, and would incorporate new information more quickly. In addition, it has recently been observed that if the simplex method is applied to the dual formulation of a subproblem, every dual-feasible extreme point visited by the simplex method has the potential to form a new necessary constraint on the preceding neighbor [HLS88].

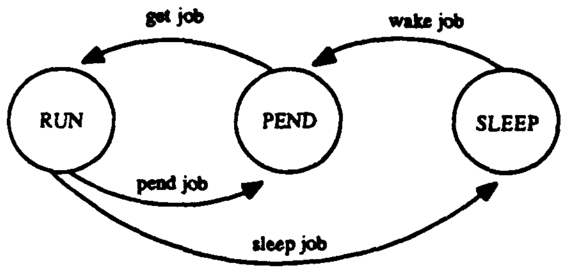

Figure 4.3. The transfer of jobs amongst queues.

The modification of a subproblem in Benders decomposition is governed by the neighbor that sent the message; if it was the preceding neighbor, then the RHS is modified; if it was the following one, then a constraint is added. Once modified, subproblems are solved using the simplex method and their solutions are broadcast to their neighbors. If a neighbor is in the **sleep** queue when the solution is sent, it is awakened and transferred to the **pend** queue, since it is now able to leave the WAIT FOR MESSAGE step (see Figure 4.3).

4.2. Reaching an Equilibrium

The p processors continue their loops, becoming idle only when there are no jobs in the **pend** queue. If this happens, a quick inspection of the **run** queue will determine if all jobs are sleeping. If so, the system is ***deadlocked***. Each processor recognizes deadlock as the signal to stop, but before stopping, one predesignated processor will execute a cleanup operation such as printing a solution. In the current design, deadlock is ***needed*** to stop the algorithm.

Given that a system cannot reach deadlock when jobs wait for messages that will always be sent, one may wonder whether the processors will ever stop, and if so, how? When a subproblem parameter modification does not cause an optimal solution of the subproblem to change, the job does ***not*** rebroadcast its solution to its neighbors. At this point, the subproblem is said to be in "equilibrium" with its neighbors; it is then placed in the sleep queue to wait for updating information. All subproblems must reach a simultaneous equilibrium for the same reason that the single-processor Benders algorithm does: 1) there are a finite number of dual extreme points in the subproblems, and 2) a different one must be communicated each time to maintain disequilibrium. Hence, at some point the collection of useful dual extreme points is exhausted. The equilibrium relationship is reflexive and transitive, and so a system-wide equilibrium is achieved.

This argument is also valid when many such dual extreme points are being passed simultaneously, as in the multiprocessor case. The condition for equilibrium is precisely deadlock, with all jobs sleeping, since no new and useful information is forthcoming.

4.3. Infeasible and Unbounded Solutions

What should be done if a subproblem finishes with an infeasible or feasible unbounded solution? If it is the first subproblem and it is infeasible (INF) then the entire SLP must be infeasible because the constraints corresponding to π_1 and ρ_1 cannot be satisfied. If it is the last subproblem finishing unbounded (UNB) then the SLP must be unbounded because there do not exist prices π_n that can satisfy the dual conditions associated with the variables x_n and w_n. These two cases correspond to the top and bottom entries in Figure 4.4. In both cases the algorithm stops.

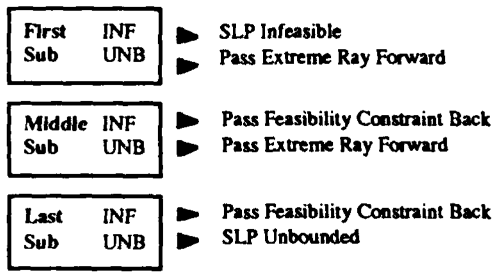

Figure 4.4. Alternative recourses for infeasible and unbounded subproblems.

In the remaining cases the algorithm continues. If a subproblem other than the first is infeasible, the infeasibility multipliers, say $\tilde{\pi}_2$, are used to impose a constraint of the form $\tilde{\pi}_2^T y_1 \geq \tilde{\pi}_2^T b_1$ on the preceding subproblem ($\tilde{\delta}_2^k$ gets a zero entry). This constraint is a necessary condition on y_1 for the feasibility of the second subproblem and thus the entire SLP. In general, a chain of such infeasibility conditions may extend from the j^{th} subproblem back to the first and indicate that the entire SLP is infeasible.

Likewise, if a subproblem other than the last finishes unbounded, the extreme point, say $\tilde{y}_1$, obtained from the primal feasible solution and the extreme ray, say $\alpha\bar{y}_1$ ($\alpha \geq 0$), from the column entering the basis and its accompanying cost $\bar{\sigma}_1$ from the incoming column's reduced cost are inserted in the following subproblem as two w_2-type columns. Only extreme-ray columns ever have coefficients in the objective row, which is why they were not explicitly included in the formulations of the previous section. The extreme-point column has its $\tilde{\delta}_1^k$ parameter set to one in order to modify the RHS (Section 3.3.) and the extreme-ray column has its $\bar{\delta}_1^k$ set to zero to allow freedom in the direction of the unboundedness. Its accompanying cost permits the subproblem to weigh the benefits of this direction against present and future costs.

If a finite solution is found to the ray-modified subproblem, the constraint it returns will necessarily restrict the old ray solution of the previous subproblem by giving the ray an unattractive positive cost.

If an infeasible solution is found to the ray-modified subproblem, the returned constraint will cut off the old ray solution outright (since it leads to infeasibility).

If an unbounded solution is found, it passes a new ray one step further. Such new rays can form a path to the last subproblem, which if unbounded too, means the entire SLP is unbounded.

Figure 4.5 summarizes the Benders algorithm for the 2-period SLP. The reader should imagine any number of *middle* subproblems inserted into the diagram to represent the general case.

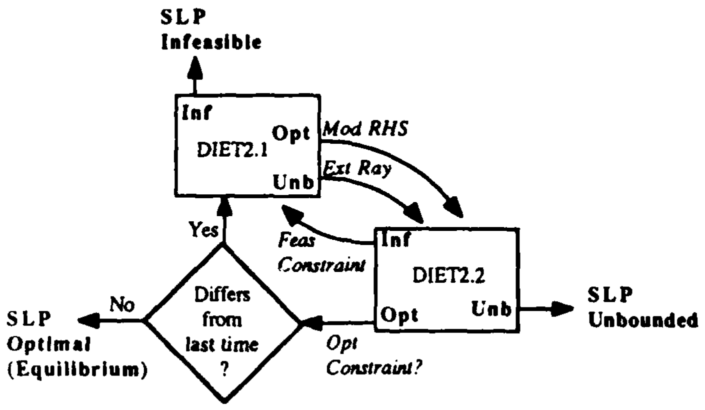

Figure 4.5. Flow of the parallel decomposition algorithm.

The boxes each represent a subproblem and its possible solution states (Inf, Unb, and Opt). The labeled arcs represent the passed information based on subproblem solutions, and the test for equilibrium is a repeated solution to the second subproblem.

5. THE FACTORS AFFECTING SPEEDUP

The behavior of the parallel decomposition algorithm will now be investigated using a seven-day diet problem (DIET7) generated in the same fashion as DIET2 was extended to DIET3. The three parameters (dimensions) of our behavioral study will be the number of subproblems, n, into which the seven-day problem is decomposed, the number of processors, p, used to solve the subproblems, and the *order* in which the subproblems are solved. In each case we will discuss the factors involved in obtaining an SLP's solution more swiftly. DIET7 turns out to be well suited for decomposition because many of the proposed algorithm's benefits are realized in the results obtained. Not all problems are so amenable to decomposition, but we feel confident that significant speedups are often attainable on parallel computers.

5.1. The Number of Subproblems

Some of the issues involved in deciding how many subproblems to create are:

1) the natural structure of the problem,
2) the overhead involved in decomposing,
3) the resulting sizes of the subproblems, or how long they will take to solve on average, and
4) the number of processors available.

The seven-period problem was solved as a single LP in 48 iterations using MINOS 5.1. This is the benchmark for comparing combinations of the above parameters. Along the subproblem dimension n, DIET7 was solved in 2-, 3-, and 7-subproblem decomposition schemes using a single processor. Efficiencies can sometimes be gained by merely breaking a problem into two subproblems because the amount of work per iteration is less and even the total work of both subproblems can be less. This proved to be true for DIET7.

As a quick measure of performance, we will assume that the amount of work per iteration is proportional to the number of rows in the subproblem—a reasonable approximation for sparse linear programs. The number of iterations per subproblem is the cumulative sum of iterations in successive solves until the entire SLP is solved. Hence, the iterative work per subproblem is approximated as the product of the number of rows and number of iterations. We can observe from Table 5.1 that the total amount of iterative work actually decreased from the single to the double subproblem case for DIET7.

p	n	sub #	# rows	# iterations	work
1	1	1	16	48	768
		totals	**16**	**48**	**768**
	2	1	11	21	231
		2	12	31	372
		totals	**23**	**52**	**603**
	3	1	9	24	216
		2	13	45	585
		3	8	34	272
		totals	**30**	**103**	**1073**
	7	1	7	28	196
		2	9	21	189
		3	9	18	162
		4	9	29	261
		5	9	24	216
		6	9	11	99
		7	6	7	42
		totals	**58**	**138**	**1165**

Table 5.1. Total work as seen in the subproblem dimension.

As the number of subproblems is increased, the overhead of reformulating increases and there is a greater need for communication. At some point, overhead and communication will begin to outweigh any benefits associated with creating more subproblems. Hence, a plot of the total work done against the number of subproblems created should look qualitatively like Figure 5.1, which attains a minimum at some point n^*.

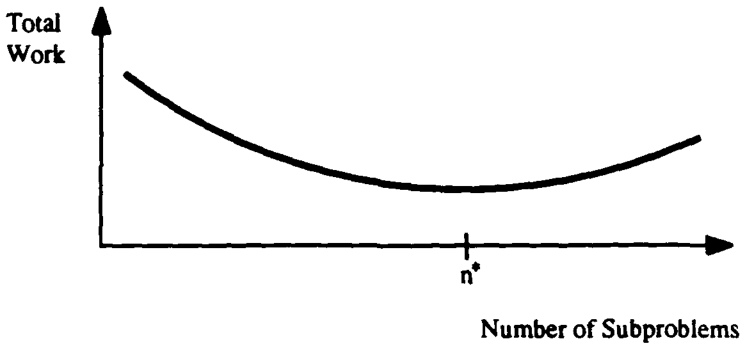

Figure 5.1. Total work in the subproblem dimension.

In the case of DIET7, $n^* = 2$ subproblems using a single processor. For a different problem or a different number of processors, the value of n^* may be different—possibly even equal to one.

5.2. The Number of Processors

Choosing the number of processors is subject to its own set of complexities. The design of any parallel algorithm is based upon the hope that the incorporation of more processors will offer almost linear speedup. However, the allocation of additional processors is a key issue because there are decreasing returns on investment. It is important that extra processors are used and do not sit idle. Having more processors than subproblems is an obvious case of inefficiency, but even when their numbers are equal, some processors will inevitably become idle (e.g. $n = p = 2$). For any given problem, there is some compromise position at which the best performance is achieved.

p	n	sub #	# rows	# iterations	work
1	7	1	7	28	196
		2	9	21	189
		3	9	18	162
		4	9	29	261
		5	9	24	216
		6	9	11	99
		7	6	7	42
		totals	**58**	**138**	**1165**
2	7	1	7	23	161
		2	9	21	189
		3	9	17	153
		4	9	19	171
		5	9	23	207
		6	9	11	99
		7	6	7	42
		totals	**58**	**121**	**1022**
4	7	1	7	23	161
		2	9	21	189
		3	9	17	153
		4	9	25	225
		5	9	22	198
		6	9	11	99
		7	6	7	42
		totals	**58**	**126**	**1067**

Table 5.2. Total work as seen in the processor dimension.

Note in Table 5.2 that with two or four processors the total work necessary to solve the problem is less than in the single processor case. This phenomenon is due to the order in which subproblems are solved, and will be discussed further in the next section.

5.3. Subproblem Ordering

In the present implementation of the parallel decomposition algorithm, there is no explicit control over the order in which subproblems are solved. They are solved as they are taken by idle processors from the **pend** queue on a first-in first-out basis. (The first in the **pend** queue will be the subproblem that received a message least recently.) The importance of order on solution time is demonstrated in Table 5.3, where DIET7 was solved twice with four processors on a Sequent Balance 8000. This machine has 8 CPUs, but because the processors are continually shared with other users, the order in which the subproblems are solved is not guaranteed to be the same in any two otherwise identical runs. As Table 5.3 shows, this can significantly affect the performance of the algorithm.

p	n	sub #	# rows	# iterations	work
4	7	1	7	23	161
		2	9	21	189
		3	9	17	153
		4	9	19	171
		5	9	23	207
		6	9	11	99
		7	6	7	42
		totals	**58**	**121**	**1022**
4	7	1	7	32	224
		2	9	26	234
		3	9	17	153
		4	9	31	279
		5	9	32	288
		6	9	11	99
		7	6	7	42
		totals	**58**	**156**	**1319**

Table 5.3. Total work as seen in the order dimension.

A possible remedy was previously alluded to during the discussion of the RUN JOB loop. If the algorithm were enhanced so that each job were put back into the **pend** queue after some predetermined number of iterations, the power of the CPUs would be more evenly distributed over the subproblems and the processors would be utilized more efficiently. This practice has the effect of incorporating new information more quickly because the latest solutions can be used to make modifications midway through a solution step. It also reduces the time that subproblems spend waiting for a processor. Future work will include such an enhancement to the algorithm.

ACKNOWLEDGEMENTS

The seed for this research into the use of parallel computers was formed three years ago in a so-called flash of awareness. By being given the opportunity to work at the Oak Ridge National Laboratory in the Autumn of 1987, where the resources of the Mathematical Sciences Section were open to me, I was able to implement this idea and see it yield new and exciting prospects. The friendly and colloquial atmosphere at the laboratory speeded my progress, and having direct access to the people and parallel computers there proved priceless. Especially valuable was the opportunity to meet visitors to the *Special Year in Numerical Linear Algebra*. Particular recognition must be given to George Dantzig, Ed Klotz, Chuck Romine, and Michael Saunders for reviewing drafts of this paper. Thanks also go to Stanford's Department of Operations Research for the use of their TEXtures software, Macintosh computers and LaserWriter printer.

TEXtures is a trademark of Addison-Wesley Publishing Company, Inc. TEX is a trademark of the American Mathematical Society. Macintosh and LaserWriter are a trademarks of Apple Computer, Inc.

REFERENCES

[Abr83] Philip G. Abrahamson (1983). A Nested Decomposition Approach for Solving Staircase Linear Programs, Ph.D. Dissertation, Dept. of Operations Research, Stanford University, Stanford, CA.

[Ben62] J. F. Benders (1962). Partitioning procedures for solving mixed-variable programming problems, *Numerische Mathematik* 4, 238–252.

[Dan63] George B. Dantzig (1963). *Linear Programming and Extensions,* Princeton University Press, Princeton.

[Chv83] Vašek Chvátal (1983). *Linear Programming,* W. H. Freeman and Company, New York and San Francisco.

[GMSW87] Philip E. Gill, Walter Murray, Michael A. Saunders and Margaret H. Wright (1987). Maintaining *LU* factors of a general sparse matrix, *Linear Algebra and its Applications*, 88/89, 239–270.

[HL81] James K. Ho and E. Loute (1981). A set of staircase linear programming test problems, *Mathematical Programming* 20, 245–250.

[HLS88] James K. Ho, Tak C. Lee and R. P. Sundarraj (1988). Decomposition of linear programs using parallel computation (revised), Invited paper at the Symposium on Parallel Optimization, Madison, WI.

[MS87] Bruce A. Murtagh and Michael A. Saunders (1987). MINOS 5.1 user's guide (revised), Report SOL 83-20R, Dept. of Operations Research, Stanford University, Stanford, CA.

[Wit83] Robert J. Wittrock (1983). Advances in a Nested Decomposition Algorithm for Solving Staircase Linear Programs, Ph.D. Dissertation, Dept. of Operations Research, Stanford University, Stanford, CA.

Appendix A. THE DIET PROBLEM SOLUTION

We wish to show how an optimal solution to (2) can be fashioned from an optimal solution of the smaller LP (1).

$$\begin{array}{rl} \text{minimize} & c^T x = z_1 \\ \pi : & Ax \geq b \\ \rho : & a^T x \geq \beta/2 \\ & x \geq 0. \end{array} \tag{1}$$

$$\begin{array}{rl} \text{minimize} & c^T x_1 + c^T x_2 = z_2 \\ \pi_1 : & Ax_1 \qquad\quad \geq b \\ \rho_1 : & a^T x_1 + a^T x_2 \geq \beta \\ \pi_2 : & \qquad\quad Ax_2 \geq b \\ & \qquad\quad x_i \geq 0 \qquad i = 1,2. \end{array} \tag{2}$$

With $\hat{x}$ and $(\hat{\pi}\ \hat{\rho})$ as the optimal primal and dual solutions to (1), and $(\hat{x}_1\ \hat{x}_2)$ and $(\hat{\pi}_1\ \hat{\rho}_1\ \hat{\pi}_2)$ as the optimal primal and dual solutions to (2), we know from strong duality that

$$\hat{z}_1 = c^T\hat{x} = b^T\hat{\pi} + \beta\hat{\rho}/2, \tag{3}$$

and

$$\hat{z}_2 = c^T\hat{x}_1 + c^T\hat{x}_2 = b^T\hat{\pi}_1 + \beta\hat{\rho}_1 + b^T\hat{\pi}_2. \tag{4}$$

a) Show that $\begin{pmatrix} \hat{x} \\ \hat{x} \end{pmatrix}$ *is primal feasible for (2).*

Clearly $\hat{x} \geq 0$. The π constraints of (1) imply that the π_1 and π_2 constraints of (2) are satisfied, and the ρ constraints of (1) imply that $a^T\hat{x} \geq \beta/2$, i.e. $2a^T\hat{x} \geq \beta$. ∎

b) Show that $(\hat{\pi}\ \hat{\rho}\ \hat{\pi})$ *is dual feasible for (2).*

The dual of (2) is

$$\begin{array}{rl} \text{maximize} & b^T\pi_1 + \beta\rho_1 + b^T\pi_2 = z_2 \\ x_1 : & A^T\pi_1 + a\rho_1 \qquad\quad \geq c \\ x_2 : & \qquad\quad a\rho_1 + A^T\pi_2 \geq c \\ & \pi_1 \geq 0, \qquad \pi_2 \geq 0. \end{array}$$

The dual of (1) implies that $A^T\hat{\pi} + a\hat{\rho} \geq c$ and $\hat{\pi} \geq 0$. ∎

c) Show that $\begin{pmatrix} \hat{x} \\ \hat{x} \end{pmatrix}$, $(\hat{\pi}\ \hat{\rho}\ \hat{\pi})$ *is optimal for (2).*

Knowing these primal feasible and dual feasible solutions of (2), it is sufficient to show that they satisfy (4). From (3),

$$2\hat{z}_1 = c^T\hat{x} + c^T\hat{x} = b^T\hat{\pi} + \beta\hat{\rho} + b^T\hat{\pi} = \hat{z}_2. \quad ∎$$

Note that if $\hat{x}$ is non-degenerate, $\begin{pmatrix} \hat{x} \\ \hat{x} \end{pmatrix}$ cannot be basic for (2) because it has an even number of variables off their bounds, and (2) has an odd number of constraints.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SOL 88-21	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Parallel Decomposition Algorithm for Staircase Linear Programs		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert Entriken		8. CONTRACT OR GRANT NUMBER(s) N00014-85-K-0343
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research - SOL Stanford University Stanford, CA 94305-4022		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 1111MA
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research - Dept. of the Navy 800 N. Quincy Street Arlington, VA 22217		12. REPORT DATE December 1988
		13. NUMBER OF PAGES 20 pages
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Linear Programming, Parallel Processors, Decomposition, Large-Scale, MINOS, Staircase

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(Please see reverse side)

A PARALLEL DECOMPOSITION ALGORITHM FOR STAIRCASE LINEAR PROGRAMS

Robert Entriken
Department of Operations Research
Stanford University

Abstract

As part of an extended research project on the parallel decomposition of linear programs, a parallel algorithm for Staircase Linear Programs was designed and implemented. This class of problems encompasses a large range of planning problems and when decomposed has simple subproblem formulations and communication patterns. This makes its solution a manageable step toward our eventual goal of producing a general code that automatically exploits problem structures of various forms.

The results presented here were derived from an implementation for a Sequent Balance 8000 shared-memory multiprocessor. The algorithm itself is message-based but can run on either shared- or distributed-memory parallel computers.

A simple diet planning problem is used to demonstrate the principles of the algorithm's development and performance. When applied to this problem, the parallel decomposition algorithm shows promise relative to present serial optimization codes. The nonlinear optimization code MINOS 5.1 is used both as a basis for comparison and as a generic subproblem solver. The greatest room for speedup is in exploiting problem structures. The results show that decomposition can improve efficiency even with a single processor. Examples are given where multiple processors lead to still greater efficiency.